On P vs. NP

Density[a]

The **density** of language $L \subseteq \Sigma^*$ is defined as

$$\text{dens}_L(n) = |\{x \in L : |\,x\,| \leq n\}|.$$

- If $L = \{0,1\}^*$, then $\text{dens}_L(n) = 2^{n+1} - 1$.
- So the density function grows at most exponentially.
- For a unary language $L \subseteq \{0\}^*$,

$$\text{dens}_L(n) \leq n + 1.$$

  - Because $L \subseteq \{\epsilon, 0, 00, \ldots, \overbrace{00 \cdots 0}^{n}, \ldots\}$.

[a] Berman and Hartmanis (1977).

Sparsity

- **Sparse languages** are languages with polynomially bounded density functions.
- **Dense languages** are languages with superpolynomial density functions.

Self-Reducibility for SAT

- An algorithm exhibits **self-reducibility** if it finds a certificate by exploiting algorithms for the *decision* version of the same problem.
- Let ϕ be a boolean expression in n variables $x_1, x_2, \ldots, x_n$.
- $t \in \{0,1\}^j$ is a **partial** truth assignment for $x_1, x_2, \ldots, x_j$.
- $\phi[\,t\,]$ denotes the expression after substituting the truth values of t for $x_1, x_2, \ldots, x_{|\,t\,|}$ in ϕ.

An Algorithm for SAT with Self-Reduction

We call the algorithm below with empty t.

```
if |t| = n then
    return φ[t];
else
    return φ[t0] ∨ φ[t1];
end if
```

The above algorithm runs in exponential time, by visiting all the partial assignments (or nodes on a depth-n binary tree).

NP-Completeness and Density[a]

Theorem 81 *If a unary language $U \subseteq \{0\}^*$ is NP-complete, then $P = NP$.*

- Suppose there is a reduction R from SAT to U.
- We shall use R to guide us in finding the truth assignment that satisfies a given boolean expression ϕ with n variables if it is satisfiable.
- Specifically, we use R to prune the exponential-time exhaustive search on p. 658.
- The trick is to keep the already discovered results $\phi[\,t\,]$ in a table H.

---

[a]Berman (1978).

```
if |t| = n then
    return φ[t];
else
    if (R(φ[t]), v) is in table H then
        return v;
    else
        if φ[t0] = "satisfiable" or φ[t1] = "satisfiable" then
            Insert (R(φ[t]), "satisfiable") into H;
            return "satisfiable";
        else
            Insert (R(φ[t]), "unsatisfiable") into H;
            return "unsatisfiable";
        end if
    end if
end if
```

The Proof (continued)

- Since R is a reduction, $R(\phi[\,t\,]) = R(\phi[\,t'\,])$ implies that $\phi[\,t\,]$ and $\phi[\,t'\,]$ must be both satisfiable or unsatisfiable.
- $R(\phi[\,t\,])$ has polynomial length $\leq p(n)$ because R runs in log space.
- As R maps to unary numbers, there are only polynomially many $p(n)$ values of $R(\phi[\,t\,])$.
- How many nodes of the complete binary tree (of invocations/truth assignments) need to be visited?
- If that number is a polynomial, the overall algorithm runs in polynomial time and we are done.

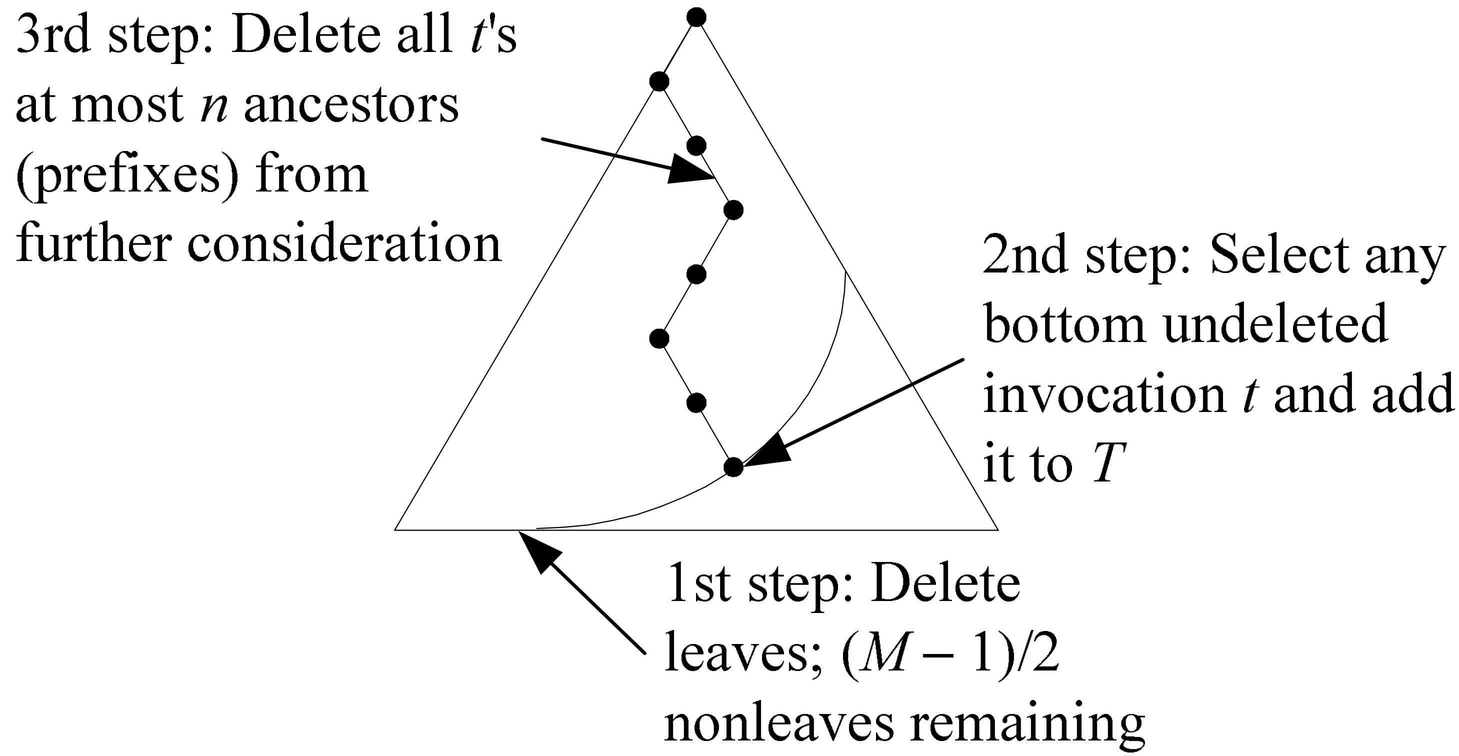
The Proof (continued)

- A search of the table takes time $O(p(n))$ in the random access memory model.
- The running time is $O(Mp(n))$, where M is the total number of invocations of the algorithm.
- The invocations of the algorithm form a binary tree of depth at most n.

The Proof (continued)

- There is a set $T = \{t_1, t_2, \ldots\}$ of invocations (partial truth assignments, i.e.) such that:

  1. $|T| \geq (M-1)/(2n)$.
  2. All invocations in T are recursive (nonleaves).
  3. None of the elements of T is a prefix of another.


3rd step: Delete all t's at most n ancestors (prefixes) from further consideration
2nd step: Select any bottom undeleted invocation t and add it to T
1st step: Delete leaves; $(M-1)/2$ nonleaves remaining

The Proof (continued)

- All invocations $t \in T$ have different $R(\phi[\,t\,])$ values.
  - None of $s, t \in T$ is a prefix of another.
  - The invocation of one started after the invocation of the other had terminated.
  - If they had the same value, the one that was invoked second would have looked it up, and therefore would not be recursive, a contradiction.
- The existence of T implies that there are at least $(M-1)/(2n)$ different $R(\phi[\,t\,])$ values in the table.

The Proof (concluded)

- We already know that there are at most $p(n)$ such values.
- Hence $(M-1)/(2n) \leq p(n)$.
- Thus $M \leq 2np(n) + 1$.
- The running time is therefore $O(Mp(n)) = O(np^2(n))$.
- We comment that this theorem holds for any sparse language, not just unary ones.[a]

[a] Mahaney (1980).

coNP-Completeness and Density

Theorem 82 (Fortung (1979)) *If a unary language $U \subseteq \{0\}^*$ is coNP-complete, then P = NP.*

- Suppose there is a reduction R from SAT COMPLEMENT to U.
- The rest of the proof is basically identical except that, now, we want to make sure a formula is unsatisfiable.

Oracles[a]

- We will be considering TMs with access to a "subroutine" or black box.
- This black box solves a language problem L (such as SAT) *in one step*.
- By presenting an input x to the black box, in one step the black box returns "yes" or "no" depending on whether $x \in L$.
- This black box is called aptly an **oracle**.

[a]Turing (1936).

Oracle Turing Machines

- A **Turing machine** $M^?$ **with oracle** is a multistring deterministic TM.
- It has a special string called the **query string**.
- It also has three special states:
  - $q?$ (the **query state**).
  - q_{yes} and q_{no} (the **answer states**).

Oracle Turing Machines (concluded)

- Let $A \subseteq \Sigma^*$ be a language.
- From $q?$, $M^?$ moves to either q_{yes} or q_{no} depending on whether the current query string is in A or not.
  - This piece of information can be used by $M^?$.
  - Think of A as a black box or a vendor-supplied subroutine.
- $M^?$ is otherwise like an ordinary TM.
- $M^A(x)$ denotes the computation of $M^?$ with oracle A on input x.

Complexity Measures of Oracle TMs

- The time complexity for oracle TMs is like that for ordinary TMs.
- Nondeterministic oracle TMs are defined in the same way.
- Let $\mathcal{C}$ be a deterministic or nondeterministic time complexity class.
- Define $\mathcal{C}^A$ to be the class of all languages decided (or accepted) by machines in $\mathcal{C}$ with access to oracle A.

An Example

- SAT COMPLEMENT $\in \text{P}^{\text{SAT}}$.
  - Reverse the answer of SAT oracle A as our answer.

```
if φ ∈ A then
    return "no"; {φ is satisfiable.}
else
    return "yes"; {φ is not satisfiable.}
end if
```

- As SAT COMPLEMENT is coNP-complete (p. 373),

$$\text{coNP} \subseteq \text{P}^{\text{SAT}}.$$

The Turing Reduction

- Recall L_1 is reducible to L_2 if there is a logspace function R such that $x \in L_1 \Leftrightarrow R(x) \in L_2$ (p. 211).
  - It is called **logspace reduction**, Karp reduction (p. 213), or **many-one reduction**.
- But the reduction in proving $L \in \mathcal{C}^A$ is more general.
  - An algorithm B for $\mathcal{C}$ with access to A exists.
  - B can call A many times within the resource bound.
  - We say L is **Turing-reducible** to A.

Two Types of Reductions

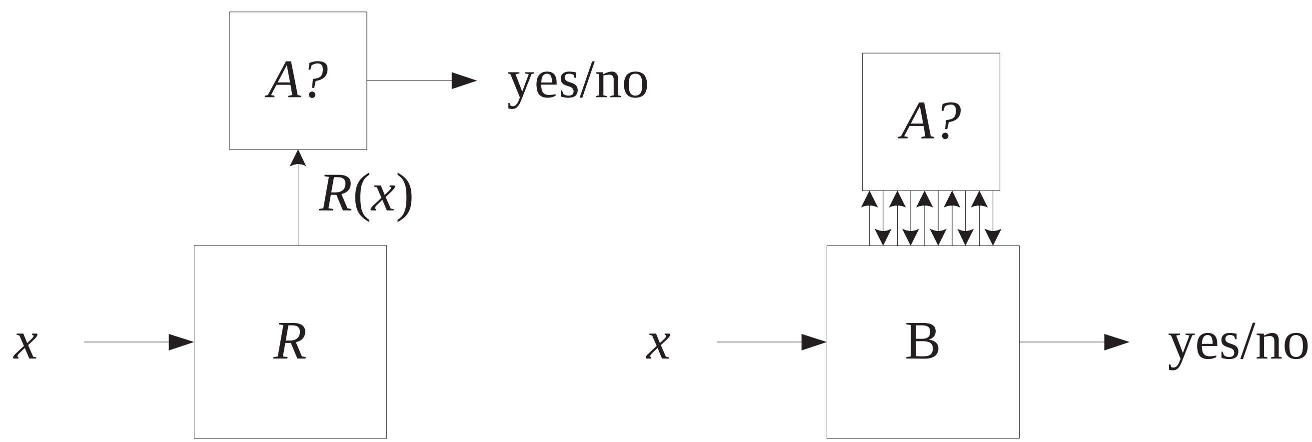
Lemma 83 *If L_1 is (logspace-) reducible to L_2, then L_1 is Turing-reducible to L_2.*

- Logspace reduction is more restrictive than Turing reduction.
- It is Turing reduction with only one query to L_2.
- Note also that a language in L also belongs in P.

Corollary 84 *If L is complete under logspace-reductions, then L is complete under Turing reductions.*

Two Types of Reductions (continued)

- Turing reduction is more general than (p. 674)—and equally valid as—logspace reduction.



- This is true even if B runs in logarithmic space and oracle A is queried only once.

Two Types of Reductions (continued)

- Turing reduction is more powerful than logspace reduction.
- For example, there are languages A and B such that A is Turing-reducible to B but not logspace-reducible to B.[a]
- However, for the class NP, no such separation has been proved.[b]

---

[a] Ladner, Lynch, and Selman (1975).

[b] If we assume NP does not have p-measure 0, then separation exists (Lutz and Mayordomo (1996)).

Two Types of Reductions (concluded)

- The Turing reduction is adaptive.
  - Later queries may depend on prior queries.
- If we restrict the Turing reduction to ask all queries before receiving any answers, the reduction is called the **truth-table reduction**.
- Separation results exist for the Turing and truth-table reductions given some conjectures.[a]

---

[a] Hitchcock and Pavan (2006).

The Power of Turing Reduction

- SAT COMPLEMENT is not likely to be reducible to SAT.
  - Otherwise, coNP = NP as SAT COMPLEMENT is coNP-complete (p. 373).
- But SAT COMPLEMENT is polynomial-time Turing-reducible to SAT.
  - SAT COMPLEMENT $\in \mathrm{P}^{\mathrm{SAT}}$ (p. 672).
  - True even though the oracle SAT is called only once!
  - The algorithm on p. 672 is *not* a logspace reduction.

Exponential Circuit Complexity

- Almost all boolean functions require $\frac{2^n}{2n}$ gates to compute (generalized Theorem 15 on p. 168).
- Progress of using circuit complexity to prove exponential lower bounds for NP-complete problems has been slow.
  - As of January 2006, the best lower bound is $5n - o(n)$.[a]

[a] Iwama and Morizumi (2002).

Exponential Circuit Complexity for NP-Complete Problems

- We shall prove exponential lower bounds for NP-complete problems using *monotone* circuits.
  - Monotone circuits are circuits without $\neg$ gates.
- Note that this does not settle the P vs. NP problem or any of the conjectures on p. 526.

The Power of Monotone Circuits

- Monotone circuits can only compute monotone boolean functions.
- They are powerful enough to solve a P-complete problem, MONOTONE CIRCUIT VALUE (p. 262).
- There are NP-complete problems that are not monotone; they cannot be computed by monotone circuits at all.
- There are NP-complete problems that are monotone; they can be computed by monotone circuits.
  - HAMILTONIAN PATH and CLIQUE.

$\text{CLIQUE}_{n,k}$

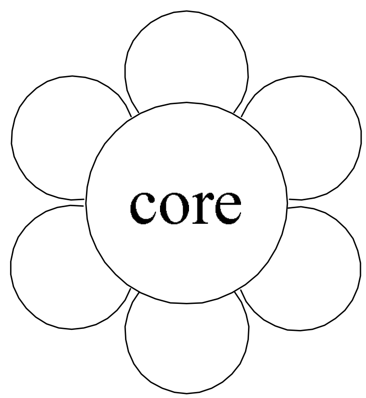
- $\text{CLIQUE}_{n,k}$ is the boolean function deciding whether a graph $G = (V, E)$ with n nodes has a clique of size k.
- The input gates are the $\binom{n}{2}$ entries of the adjacency matrix of G.
  - Gate g_{ij} is set to true if the associated undirected edge $\{ i, j \}$ exists.
- $\text{CLIQUE}_{n,k}$ is a monotone function.
- Thus it can be computed by a monotone circuit.
- This does not rule out that nonmonotone circuits for $\text{CLIQUE}_{n,k}$ may use fewer gates.

Crude Circuits

- One possible circuit for $\text{CLIQUE}_{n,k}$ does the following.
  1. For each $S \subseteq V$ with $|S| = k$, there is a subcircuit with $O(k^2)$ $\wedge$-gates testing whether S forms a clique.
  2. We then take an OR of the outcomes of all the $\binom{n}{k}$ subsets $S_1, S_2, \ldots, S_{\binom{n}{k}}$.
- This is a monotone circuit with $O(k^2\binom{n}{k})$ gates, which is exponentially large unless k or $n-k$ is a constant.
- A **crude circuit** $\text{CC}(X_1, X_2, \ldots, X_m)$ tests if *any* of $X_i \subseteq V$ forms a clique.
  - The above-mentioned circuit is $\text{CC}(S_1, S_2, \ldots, S_{\binom{n}{k}})$.

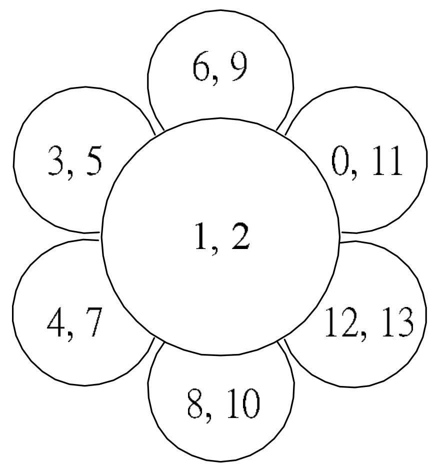
Sunflowers

- Fix $p \in \mathbb{Z}^+$ and $\ell \in \mathbb{Z}^+$.
- A **sunflower** is a family of p sets $\{P_1, P_2, \ldots, P_p\}$, called **petals**, each of cardinality at most ℓ.
- All pairs of sets in the family must have the same intersection (called the **core** of the sunflower).



A Sample Sunflower

$$\{\{1, 2, 3, 5\}, \{1, 2, 6, 9\}, \{0, 1, 2, 11\},$$
$$\{1, 2, 12, 13\}, \{1, 2, 8, 10\}, \{1, 2, 4, 7\}\}$$



The Erdős-Rado Lemma

Lemma 85 *Let $\mathcal{Z}$ be a family of more than $M = (p-1)^\ell \ell!$ nonempty sets, each of cardinality ℓ or less. Then $\mathcal{Z}$ must contain a sunflower (of size p).*

- Induction on ℓ.
- For $\ell = 1$, p different singletons form a sunflower (with an empty core).
- Suppose $\ell > 1$.
- Consider a *maximal* subset $\mathcal{D} \subseteq \mathcal{Z}$ of *disjoint* sets.
  - Every set in $\mathcal{Z} - \mathcal{D}$ intersects some set in $\mathcal{D}$.

The Proof of the Erdős-Rado Lemma (continued)

- Suppose $\mathcal{D}$ contains at least p sets.
  - $\mathcal{D}$ constitutes a sunflower with an empty core.
- Suppose $\mathcal{D}$ contains fewer than p sets.
  - Let C be the union of all sets in $\mathcal{D}$.
  - $|\,C\,| \leq (p-1)\ell$ and C intersects every set in $\mathcal{Z}$.
  - There is a $d \in C$ that intersects more than $\frac{M}{(p-1)\ell} = (p-1)^{\ell-1}(\ell-1)!$ sets in $\mathcal{Z}$.
  - Consider $\mathcal{Z}' = \{Z - \{d\} : Z \in \mathcal{Z}, d \in Z\}$.
  - $\mathcal{Z}'$ has more than $M' = (p-1)^{\ell-1}(\ell-1)!$ sets.

The Proof of the Erdős-Rado Lemma (concluded)

- (continued)
  - M' is just M with ℓ decreased by one.
  - $\mathcal{Z}'$ contains a sunflower by induction, say
    $$\{P_1, P_2, \ldots, P_p\}.$$
  - Now,
    $$\{P_1 \cup \{d\}, P_2 \cup \{d\}, \ldots, P_p \cup \{d\}\}$$
    is a sunflower in $\mathcal{Z}$.

Comments on the Erdős-Rado Lemma

- A family of more than M sets must contain a sunflower.
- **Plucking** a sunflower entails replacing the sets in the sunflower by its core.
- By repeatedly finding a sunflower and plucking it, we can reduce a family with more than M sets to a family with at most M sets.
- If $\mathcal{Z}$ is a family of sets, the above result is denoted by pluck($\mathcal{Z}$).

An Example of Plucking

- Recall the sunflower on p. 685:

$$\begin{aligned}\mathcal{Z} &= \{\{1,2,3,5\},\{1,2,6,9\},\{0,1,2,11\},\\ &\quad \{1,2,12,13\},\{1,2,8,10\},\{1,2,4,7\}\}\end{aligned}$$

- Then

$$\text{pluck}(\mathcal{Z}) = \{\{1,2\}\}.$$

Razborov's Theorem

Theorem 86 (Razborov (1985)) *There is a constant c such that for large enough n, all monotone circuits for* $\text{CLIQUE}_{n,k}$ *with $k = n^{1/4}$ have size at least $n^{cn^{1/8}}$.*

- We shall approximate any monotone circuit for $\text{CLIQUE}_{n,k}$ by a restricted kind of crude circuit.
- The approximation will proceed in steps: one step for each gate of the monotone circuit.
- Each step introduces few errors (false positives and false negatives).
- But the resulting crude circuit has exponentially many errors.

Alexander Razborov (1963–)

The Proof

- Fix $k = n^{1/4}$.
- Fix $\ell = n^{1/8}$.
- Note that
$$2\binom{\ell}{2} \leq k.$$
- p will be fixed later to be $n^{1/8} \log n$.
- Fix $M = (p-1)^{\ell} \ell!$.
  - Recall the Erdős-Rado lemma (p. 686).

The Proof (continued)

- Each crude circuit used in the approximation process is of the form $\mathrm{CC}(X_1, X_2, \ldots, X_m)$, where:
  - $X_i \subseteq V$.
  - $|X_i| \leq \ell$.
  - $m \leq M$.
- We shall show how to approximate any circuit for $\textsc{clique}_{n,k}$ by such a crude circuit, inductively.
- The induction basis is straightforward:
  - Input gate g_{ij} is the crude circuit $\mathrm{CC}(\{i, j\})$.

The Proof (continued)

- Any monotone circuit can be considered the OR or AND of two subcircuits.
- We shall show how to build approximators of the overall circuit from the approximators of the two subcircuits.
  - We are given two crude circuits $\mathrm{CC}(\mathcal{X})$ and $\mathrm{CC}(\mathcal{Y})$.
  - $\mathcal{X}$ and $\mathcal{Y}$ are two families of at most M sets of nodes, each set containing at most ℓ nodes.
  - We construct the approximate OR and the approximate AND of these subcircuits.
  - Then show both approximations introduce few errors.

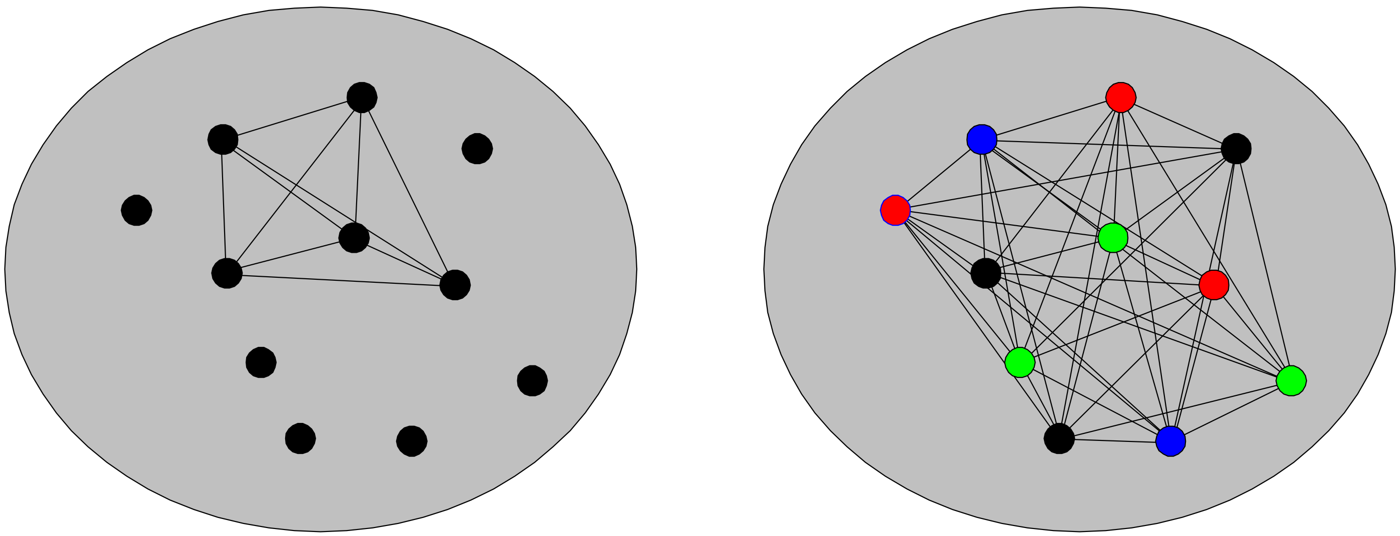
The Proof: Positive Examples

- Error analysis will be applied to only **positive examples** and **negative examples**.
- A positive example is a graph that has $\binom{k}{2}$ edges connecting k nodes in all possible ways.
- There are $\binom{n}{k}$ such graphs.
- They all should elicit a true output from $\text{CLIQUE}_{n,k}$.

The Proof: Negative Examples

- Color the nodes with $k-1$ different colors and join by an edge any two nodes that are colored differently.
- There are $(k-1)^n$ such graphs.
- They all should elicit a false output from $\text{CLIQUE}_{n,k}$.
  - Each set of k nodes must have 2 identically colored nodes; hence there is no edge between them.

Positive and Negative Examples with $k = 5$



A positive example   A negative example

The Proof: OR

- $\mathrm{CC}(\mathcal{X} \cup \mathcal{Y})$ is *equivalent to* the OR of $\mathrm{CC}(\mathcal{X})$ and $\mathrm{CC}(\mathcal{Y})$.
- Violations occur when $|\mathcal{X} \cup \mathcal{Y}| > M$.
- Such violations can be eliminated by using

$$\mathrm{CC}(\mathrm{pluck}(\mathcal{X} \cup \mathcal{Y}))$$

  as the approximate OR of $\mathrm{CC}(\mathcal{X})$ and $\mathrm{CC}(\mathcal{Y})$.
- We now count the numbers of errors this approximate OR makes on the positive and negative examples.

The Proof: OR (concluded)

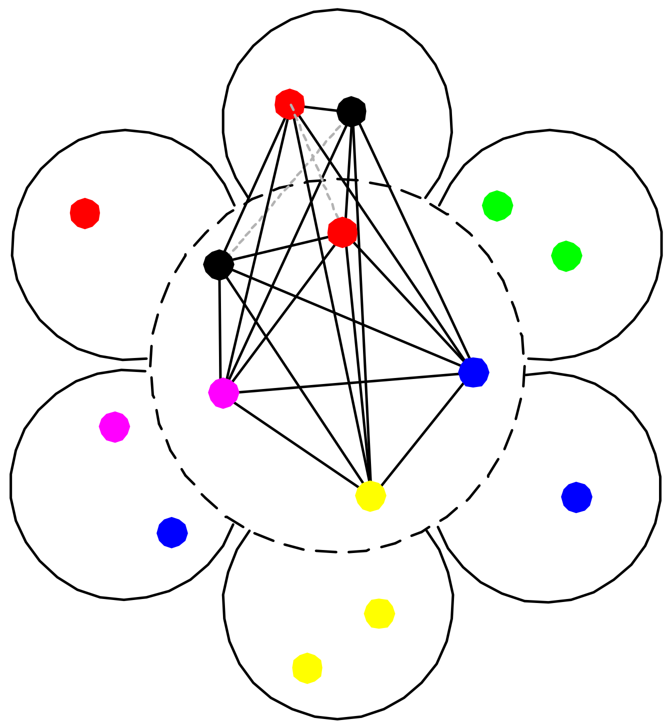
- CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) *introduces* a **false positive** if a negative example makes both CC($\mathcal{X}$) and CC($\mathcal{Y}$) return false but makes CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) return true.
- CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) *introduces* a **false negative** if a positive example makes either CC($\mathcal{X}$) or CC($\mathcal{Y}$) return true but makes CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) return false.
- How many false positives and false negatives are introduced by CC(pluck($\mathcal{X} \cup \mathcal{Y}$))?

The Number of False Positives

Lemma 87 CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) *introduces at most* $\frac{M}{p-1} 2^{-p}(k-1)^n$ *false positives.*

- A plucking replaces the sunflower $\{Z_1, Z_2, \ldots, Z_p\}$ with its core Z.
- A false positive is *necessarily* a coloring such that:
  - There is a pair of identically colored nodes in each petal Z_i (and so both crude circuits return false).
  - But the core contains distinctly colored nodes.
    - This implies at least one node from each same-color pair was plucked away.
- We now count the number of such colorings.

Proof of Lemma 87 (continued)



Proof of Lemma 87 (continued)

- Color nodes V at random with $k-1$ colors and let $R(X)$ denote the event that there are repeated colors in set X.
- Now $\text{prob}[R(Z_1) \wedge \cdots \wedge R(Z_p) \wedge \neg R(Z)]$ is at most

$$\begin{aligned} & \text{prob}[R(Z_1) \wedge \cdots \wedge R(Z_p) | \neg R(Z)] \\ = & \prod_{i=1}^{p} \text{prob}[R(Z_i) | \neg R(Z)] \leq \prod_{i=1}^{p} \text{prob}[R(Z_i)]. \quad (11) \end{aligned}$$

  - First equality holds because $R(Z_i)$ are independent given $\neg R(Z)$ as Z contains their only common nodes.
  - Last inequality holds as the likelihood of repetitions in Z_i decreases given no repetitions in $Z \subseteq Z_i$.

Proof of Lemma 87 (continued)

- Consider two nodes in Z_i.
- The probability that they have identical color is $\frac{1}{k-1}$.
- Now $\text{prob}[\, R(Z_i)\,] \leq \frac{\binom{|Z_i|}{2}}{k-1} \leq \frac{\binom{\ell}{2}}{k-1} \leq \frac{1}{2}$.
- So the probability[a] that a random coloring is a new false positive is at most 2^{-p} by inequality (11).
- As there are $(k-1)^n$ different colorings, each plucking introduces at most $2^{-p}(k-1)^n$ false positives.

---

[a]Proportion, i.e.

Proof of Lemma 87 (concluded)

- Recall that $|\mathcal{X} \cup \mathcal{Y}| \leq 2M$.
- Each plucking reduces the number of sets by $p - 1$.
- Hence at most $\frac{M}{p-1}$ pluckings occur in pluck$(\mathcal{X} \cup \mathcal{Y})$.
- At most
$$\frac{M}{p-1}\, 2^{-p}(k-1)^n$$
false positives are introduced.

The Number of False Negatives

Lemma 88 CC(pluck($\mathcal{X} \cup \mathcal{Y}$)) *introduces no false negatives.*

- Each plucking replaces a set in a crude circuit by a subset.
- This makes the test less stringent.
  - For each $Y \in \mathcal{X} \cup \mathcal{Y}$, there must exist at least one $X \in \text{pluck}(\mathcal{X} \cup \mathcal{Y})$ such that $X \subseteq Y$.
  - So if Y is a clique, then this X is also a clique.
- So plucking can only increase the number of accepted graphs.

The Number of False Negatives (concluded)